Izvještaj o korištenim algoritmima za pronalazak optimalne rute

1. Uvod

Ovaj izvještaj pruža detaljan pregled algoritamskog rješenja implementiranog u klasi **RouteFinder** za pronalazak optimalnih ruta u transportnoj mreži. Cilj je bio razviti fleksibilan sistem koji može pronaći najbolje putanje na osnovu različitih kriterijuma, kao što su **vrijeme putovanja**, **cijena karte** i **broj presjedanja**. Umjesto pronalaska samo jedne, apsolutno optimalne rute, algoritam je dizajniran da pronađe i rangira top N najboljih ruta, pružajući korisniku više opcija, kao što sam zadatak i nalaže.

2. Odabrani algoritam i razlog odabira

Za pronalazak ruta, odabran je **modifikovani Dijkstrin algoritam**.

Dijkstrin algoritam je idealan za pronalaženje najkraćih putanja u grafovima sa ne-negativnim težinama, što je ovdje slučaj (vrijeme, cijena i presjedanja su uvijek pozitivne vrijednosti). Njegova osnovna verzija pronalazi samo jednu, apsolutno najbolju rutu. Međutim, naš problem zahtijeva pronalaženje više (top N) optimalnih ruta.

Stoga je osnovni Dijkstrin algoritam modifikovan na sljedeći način:

1. **Skladištenje više putanja po čvoru:** Umjesto da smješta samo jednu najbolju putanju do svake stanice, algoritam sada čuva listu K najboljih putanja za svaku stanicu (u našem slučaju K je limit koji, tehnički, i ne mora da postoji, međutim ključan je zbog performansi algoritma i smanjenja memorijske potrošnje).
2. **Upotreba prioritetnog reda (PriorityQueue):** Prioritetni red je ključna struktura podataka u ovom algoritmu. On sortira čvorove koje treba posjetiti na osnovu trenutnog "troška" putovanja, osiguravajući da se uvijek prvo istražuju najperspektivnije rute.
3. **Nastavak pretrage nakon pronalaska cilja:** Za razliku od standardnog Dijkstrinog algoritma koji se zaustavlja čim dođe do cilja, naš algoritam nastavlja pretragu kako bi istražio i pronašao i druge, potencijalno manje optimalne, ali i dalje relevantne rute.

Ova modifikacija nam omogućava da efikasno pronađemo top N rute koje ispunjavaju zahtjeve, bez značajnog uticaja na performanse u poređenju sa pokretanjem više pretraga.

3. Kriterijumi optimizacije i logika poređenja

Algoritam je dizajniran da fleksibilno radi sa tri glavna kriterijuma:

- **Vrijeme putovanja (time):** Prioritet ima putanja sa najkraćim ukupnim vremenom, uključujući i vrijeme čekanja na presjedanja.
- **Cijena (price):** Primarni cilj je pronaći najjeftiniju putanju.

- **Broj presjedanja (transfers):** Algoritam se fokusira na putanje sa najmanjim brojem presjedanja.

Da bi se osigurala konzistentnost, stvoren je jedinstveni komparator (*comparePaths* metoda u klasi *RouteFinder*) koji dinamički mijenja primarni kriterijum poređenja. Ako su dvije rute jednake po primarnom kriterijumu, algoritam koristi sekundarne kriterijume (npr. ako su dva puta jednako duga, bira se jeftiniji) kako bi razriješio izjednačenje.

Sekundarni kriterijumi:

- **time (vrijeme):** *vrijeme* (primarno) > *cijena* > *presjedanja*
- **price (cijena):** *cijena* (primarno) > *vrijeme* > *presjedanja*
- **transfers (presjedanja):** *presjedanja* (primarno) > *vrijeme* > *cijena*

Ovaj pristup omogućava da se prikažu zaista najbolje opcije, čak i ako se primarni kriterijumi poklapaju.

4. Strukture podataka

Za implementaciju algoritma korištene su sljedeće ključne klase:

- **NodeState:** Pomoćna klasa koja predstavlja stanje pretrage. Sadrži referencu na trenutnu stanicu, vrijeme dolaska i kumulativnu putanju do te tačke. *NodeState* implementira *Comparable* interfejs, što omogućava *PriorityQueue* da ga pravilno sortira na osnovu definisanih kriterijuma optimizacije.
- **Path:** Klasa koja predstavlja kompletnu rutu. Sadrži listu **RouteSegment** objekata i kumulativne vrijednosti (ukupna cijena, vrijeme putovanja, broj presjedanja). Pruža sve potrebne podatke za prikaz rute i za poređenje u algoritmu.
- **RouteSegment:** Osnovni gradivni blok putanje, koji predstavlja jednu dionicu putovanja od stanice do stanice. Sadrži detalje o polasku, vremenima i cijenama.

5. Zaključak

Modifikovani Dijkstrin algoritam, implementiran u klasi *RouteFinder*, uspješno rješava problem pronalaska i rangiranja top N optimalnih ruta u transportnoj mreži. Njegova fleksibilnost u pogledu kriterijuma optimizacije, u kombinaciji sa efikasnim korištenjem struktura podataka kao što su *PriorityQueue* i prilagođene klase *NodeState* i *Path*, čini ga pouzdanim rješenjem za datu problematiku. Krajnji rezultat je sistem koji ne samo da pronalazi najbržu ili najjeftiniju putanju, već pruža i korisne alternative koje odgovaraju različitim potrebama korisnika.